ABB Project : Data Driven Modeling for Control of Industrial Systems

Group Members:

Karl Jonsson Jezdimir Milošević Alexandros Nikou*

Winston Garcia-Gabin Xiaoming Hu Kateryna Mishchenko

Lei Feng

September 7, 2018

Contents

*The PhD students, and responsible for the report, are with the KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden. Email: {karljo, jezdimir, anikou}@kth.se.

1 Introduction

Control systems are very important for modern day society. These systems are used to operate various processes, ranging from microbots in medicine to large scale infrastructures such as power networks. By using control algorithms, these processes can be operated more efficiently and reliably.

In modern day control theory, optimal control algorithms are mostly developed based on a model of physical process being controlled. One approach to obtain the model could be by using a physical laws governing behavior of the process. However, in some cases, the process can be too complicated to use this approach. In that case, data driven modeling can be applied. In this type of modeling, the structure of the model is assumed, and model parameters are then tuned to match data collected from the process. In recent years, advances in artificial intelligence provided control engineers with efficient and robust algorithms for data based model building. For example, model can be obtained by using neural networks or regression tree based methods.

In this report, we are interested in fitting different types of models for multiple input multiple output thrust propulsion system of ABB company. In particular, the following tasks were specified by the company.

1. The first task is to create a model that maps inputs of the system to the outputs. To accomplish this task, we are given the data collected from the operation of a propulsion system. Additionally, machine learning algorithms should be used to fit the model that matches this data.

2. Secondly, to develop the optimal control techniques, the model that maps outputs to inputs is of big interest. However, this task may be more troublesome, since different inputs values can be mapped to the same outputs. Thus, non-injectivity property should be investigated prior to fitting inverse models.

3. The final task is to investigate robustness of the models in respect to quality of the data we use for training. In particular, we are interested to answer if corrupting the data with noise can considerably influence quality of models returned by the machine learning algorithms we used.

The remainder of the document explains how we tackled these tasks, and is organized as follows. In Section 2, we introduce the process we would like to fit, and possible modeling architectures. In Section 3 we formulate the problem, and in Section 4, we introduce the tools for solving the problem. In Sections 5-Section 7 we tackle the tasks one to three, respectively. Finally, in Section 8, we discuss the problems we considered, summarize conclusions, and introduce possible directions for the future work.

2 Propulsion System: System Description and Possible Model Architectures

2.1 Propulsion System

The process we consider has three inputs (u_1, u_2, u_3) and four outputs (y_1, y_2, y_3, y_4). Input u_1 represents trajectory of a propeller, u_2 is rotational speed, and u_3 stands for the angle that defines trajectory of a vessel. Output y_1 is performance output, and y_4 is torque. Nature of outputs y_3 and y_4 was not revealed due to company policy.

Initially we plotted the given data to be able to have a first overview of how the required function $f(\cdot)$ looks. Figures 2-9 show how the output y_1-y_4 are related with the inputs u_1, u_2 and u_3. By observing the aforementioned figures, we reach to the following conclusions:

- The outputs y_1-y_4 have data concentrated in 3 bands, according to the values of u_3. Band 1: $-2 \leq u_3 \leq 2$; band 2: $-6.8 \leq u_3 \leq -4$; and band 3: $4 \leq u_3 \leq 7.1$.
- Outside of the range of these bands, no data were given.
- The output y_1, which is a performance measure, is maximized in the band where $-2 \leq u_3 \leq 2$. It should be noted that a desired optimal performance is achieved when $y_1 \to 1$.
- There are a few negative data of the output y_1 in the band where $-2 \leq u_3 \leq 2$.

Thus, in the experimental testing that will be presented hereafter, we choose to concentrate our analysis in the important band in which it holds that $-2 \leq u_3 \leq 2$.

By examining the given data, it yields that the outputs are in the range

$$\begin{aligned} -14.5 \leq y_1 \leq 0.9, \qquad & -2.9 \times 10^5 \leq y_2 \leq 1.9 \times 10^5 \\ -3.3 \times 10^5 \leq y_3 \leq 3.1 \times 10^5, \qquad & 5 \times 10^3 \leq y_4 \leq 1.1 \times 10^6. \end{aligned}$$

2.2 Possible Model Architectures

As we mentioned, our task is to build a system model and inverse model. Here we discuss possible model architectures that can be adopted. The discussion is about building an original model, but the same arguments hold for the inverse model.

Architectures are shown in Figure 1. In the first architecture shown in Figure 1 (a), we simply adopt a multiple input multiple output model. Within control theory, these types of models are in the most cases adopted, and tools for estimating them are well developed. However, this model architecture have more sense to adopt once we have a coupling between outputs. Additionally, in case that one of the outputs is much larger than the other, which is the case with our system, that output can have much larger influence to overall error. Thus, the algorithm may focus all its efforts in trying to fit one of the outputs, while the quality of fit for other outputs may be neglected. To avoid this problem, in some of the experiments we performed pre-scaling as follows

$$u_i = \frac{u_i - \bar{u}}{u_{\max}}, \qquad y_i = \frac{y_i - \bar{y}}{y_{\max}},$$

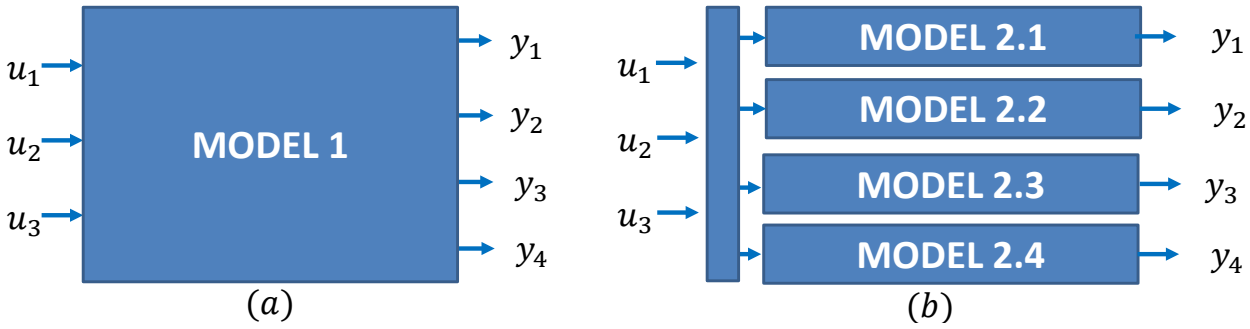

Figure 1: Possible model architectures that can be adopted.

where

$$\bar{\chi} = \frac{1}{N}\sum_{i=1}^{N}\chi_i, \qquad \chi_{\max} = \max_i |\chi_i - \bar{\chi}|, \qquad \chi \in \{u, y\}.$$

This results in inputs u_i and outputs y_i that are within the bounds $[-1, 1]$.

However, since our model represents a static mapping (each output is only influenced by the inputs), it is also reasonable to construct one model each output, as shown in Figure 1 (b). The reasons for adopting this model are twofold. Firstly, since we are fitting only a single output, we do not need to worry about scaling. Moreover, some of the algorithms we considered were developed for fitting only multiple input single output models, so this model architecture is the only solution in that case.

2.2.1 Measuring Quality of Models

Finally, we disscuss how we measure quality of the models. Assume we have obtained an approximation $\hat{y}$ of an output y. Quality of $\hat{y}$ can be measured in several ways. The first one is to simply calculate the *estimation error*

$$e = y - \hat{y}$$

which is the difference between the exact value y and the approximation $\hat{y}$. However, once we need to compare the quality of estimation for several outputs of different orders of magnitude, it makes more sense to use relative error

$$e_{\text{rel}} := \frac{y - \hat{y}}{|y|}$$

which is estimation error scaled by magnitude of y. The proble with the relative error is that it tend to be sensitive if the values of $|y|$ are close to zero. This was the case with our dataset, so to overcome this issue, the following measure is used instead of relative error

$$e_{\text{rob}} := \frac{y - \hat{y}}{\max\{|y|, |\hat{y}|\}} \tag{1}$$

Since we had a large number of measurements, we adopted the mean value and the variance of e_{rob}, and used it to measure the quality of our models.

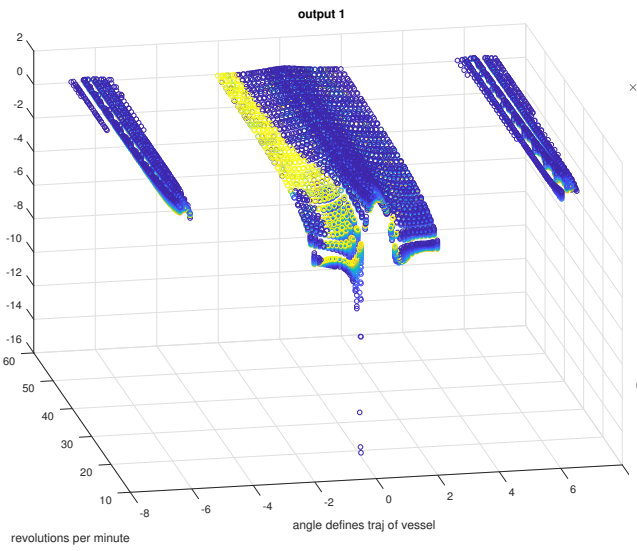

Figure 2: Plot of y_1 with respect to u_3 (right axis) and u_2 (left axis)

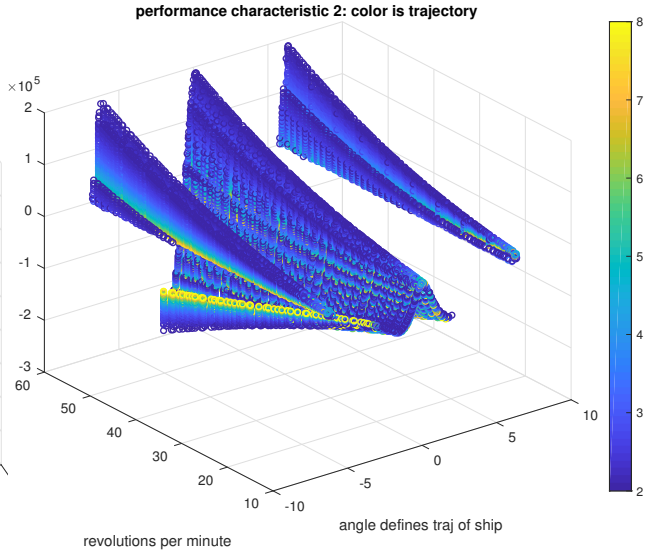

Figure 3: Plot of y_2 with respect to u_3 (right axis) and u_2 (left axis)

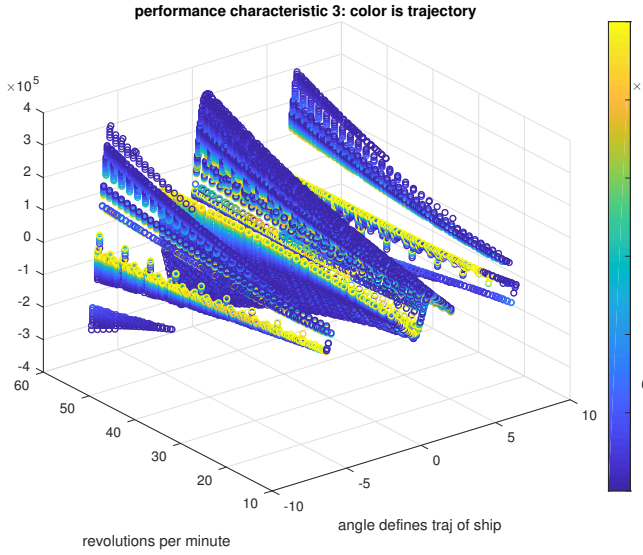

Figure 4: Plot of y_3 with respect to u_3 (right axis) and u_2 (left axis)

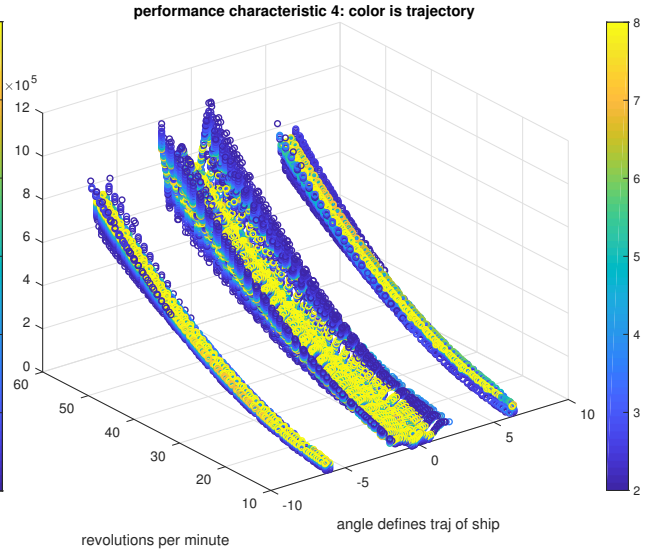

Figure 5: Plot of y_4 with respect to u_3 (right axis) and u_2 (left axis)

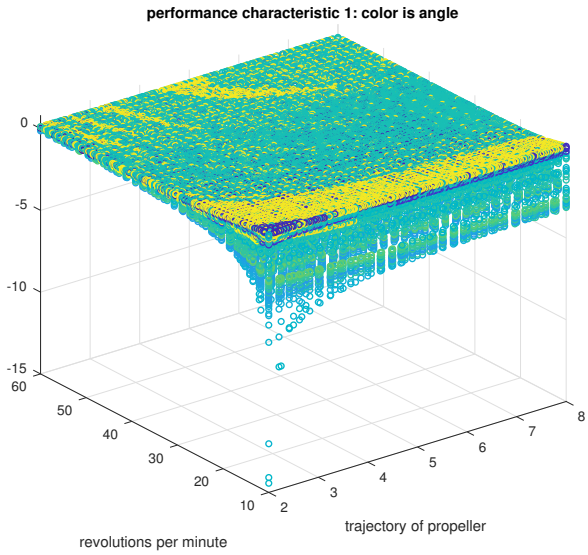

Figure 6: Plot of y_1 with respect to u_1 (right axis) and u_2 (left axis)

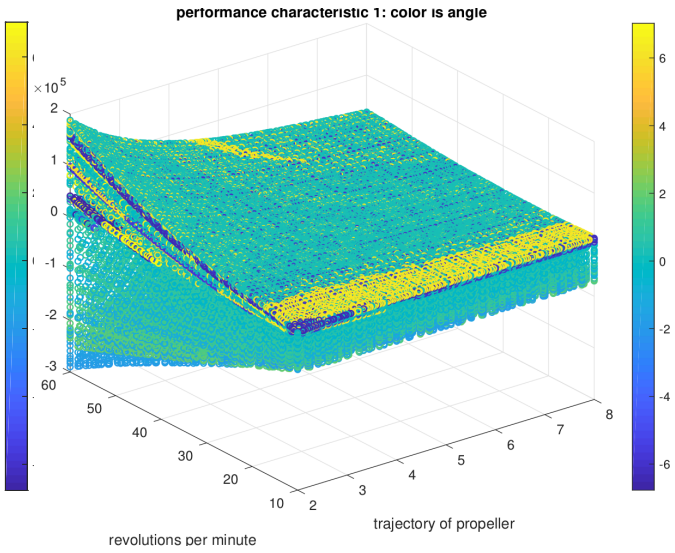

Figure 7: Plot of y_2 with respect to u_1 (right axis) and u_2 (left axis)

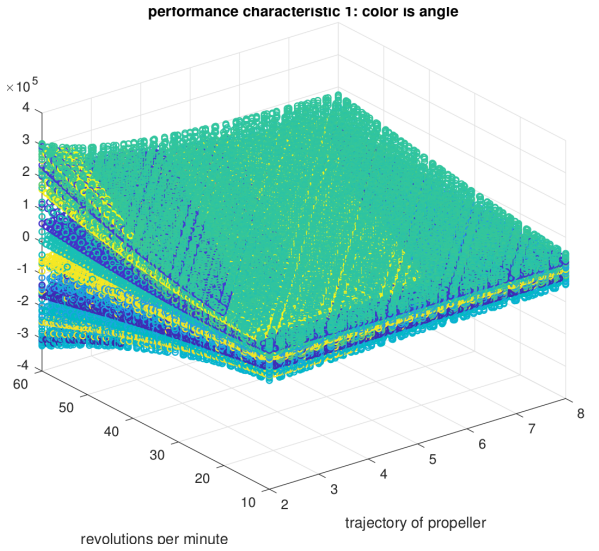

Figure 8: Plot of y_3 with respect to u_1 (right axis) and u_2 (left axis)

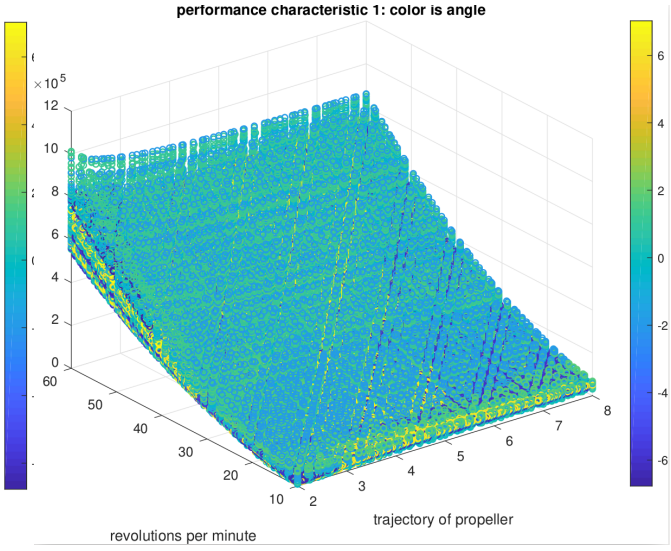

Figure 9: Plot of y_4 with respect to u_1 (right axis) and u_2 (left axis)

3 Problem Formulation

We are given the input $\mathbf{u} = (u_i)_{i=1,2,3} \in \mathbb{R}^3$ and corresponding output data $\mathbf{y} = (y_i)_{i=1,\dots,4} \in \mathbb{R}^4$ of a propulsion system for Npoints. The current data-set has three inputs and four outputs.

During the workshop the data-set which is considered $(\mathbf{u}_n, \mathbf{y}_n)$ for $n = 1, \dots, N$ was generated by an analytically derived model $\mathbf{y} = f_{\text{analytic}}(\mathbf{u})$ of the propulsion system. The analytical model is based on first principles. In the present the analytical model is imperfect in the sense that it contains some modeling error stemming from the fact that all physical factors influencing the system has not been considered. For instance, effects of turbulence and the interaction of the propulsion system and the hull of the boat is not captured by the analytical model.

The problem formulation was to use the generated data and derive a data driven model based on this system. The long term goal of the project for ABB is for the data to come from actual measurements of the real propulsion system and in this fashion circumvent the dependence on the imperfect analytical model.

3.1 Regression Problem

The *regression problem* deals with the question of learning from data in which there is some pattern. In the present case we are dealing with a problem in the class of *supervised learning problems*. This means that given a set of inputs $\mathbf{u} \in \mathscr{X} = \mathbb{R}^d$ to which we are given the corresponding desired outputs $\mathbf{y} \in \mathscr{Y} = \mathbb{R}^\ell$ of the model. One way to view the problem is that we are looking for a function $f_{\text{reg}} : \mathscr{X} \to \mathscr{Y}$ such that $\mathbf{y}_n \approx f_{\text{reg}}(\mathbf{u}_n)$ for all $n = 1, \dots, N$. The space of functions in which we are looking for f_{reg} is denoted by $\mathscr{H}$ and referred to as the *hypothesis space*.

This type of problem stands in contrast to an *unsupervised learning problem* where only inputs $\mathbf{u}$ are given (without any labels $\mathbf{y}$); an example of such a problem is the k-means clustering problem where we want to split the data set $\{\mathbf{u}_n\}_{n=1,\dots,N}$ in to k clusters $C_1, \dots, C_k$ such the sum of the variances of each cluster is minimized.

Depending on the characteristics of the given data $(\mathbf{u}_n, \mathbf{y}_n)$ together with the intended usage of the regression model f_{reg} the optimal choice of the hypothesis space $\mathscr{H}$ to search for the approximation f_{reg} in may vary. For instance, certain sets of data and certain spaces $\mathscr{H}$ together with some training criteria guarantees a one-step *training process*. For instance suppose that our model is $f_{\text{reg}} : \mathbb{R}^3 \to \mathbb{R}$ and assumed to be linear, i.e. $f_{\text{reg}}(\mathbf{u}) = a_1u_1 + a_2u_2 + a_3u_3$ for some choice of $\mathbf{a} = (a_1, a_2, a_3)^t$. We say that f_{reg} is parameterized by the vector $\mathbf{a}$ and write $f_{\text{reg}} = f_{\text{reg}}(\cdot; \mathbf{a})$ in order to highlight the parameter dependence. Given N data points we have the desired equalities

$$\begin{pmatrix} u_1^1 & u_2^1 & u_3^1 \\ u_1^2 & u_2^2 & u_3^2 \\ \vdots & \vdots & \vdots \\ u_1^N & u_2^N & u_3^N \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \Leftrightarrow U\mathbf{a} = \mathbf{y}$$

then the choice of $\mathbf{a}$ which minimizes $\|U\mathbf{a} - \mathbf{y}\|_2$ can be expressed using the pseudo-inverse of U, denoted by $U^\dagger$, according to

$$\mathbf{a} = U^\dagger \mathbf{y} \text{ where } U^\dagger := (U^tU)^{-1}U^t.$$

The general structure of the supervised learning problem is that we are given a set of data points $(\mathbf{u}_n, \mathbf{y}_n) \in \mathscr{X} \times \mathscr{Y}$ to which we wish to find some function $f_{\text{reg}} : \mathscr{X} \to \mathscr{Y}$ which maps inputs to outputs, where f_{reg} is taken to be in the space $\mathscr{H}$. In the example above the measure of the fit was quantified by finding the parameter vector $\mathbf{a}$ such that $\|U\mathbf{a} - \mathbf{y}\|_2$ (the so called *error measure*) was as small as possible. The process of finding the optimal choice of parameters $\mathbf{a}$ will be referred to as the *learning algorithm* (which in the example above, $\mathbf{a} = U^{\dagger}\mathbf{y}$, could be characterized by an analytical). Certain learning algorithms can be achieved in a one-step fashion, others are of an iterative kind where one usually refer to one step in the training process as one *epoch*.

Suppose that we fix a regression function $f \in \mathscr{H}$. The *in sample error* E_{in} is the error that f gives on the data which was provided to the learning algorithm, i.e. data that has been "seen" during the training process. One commonly used error-measure is the *mean squared error*. For a given regression model f we define the mean squared in sample error to be

$$E_{\text{in,mse}}(f) = \frac{1}{N}\sum_{n=1}^{N} \|f_{\text{reg}}(\mathbf{x}_n) - \mathbf{y}_n\|_2^2. \tag{2}$$

The *out of sample error* E_{out} is the error which the function f will yield when encountering new data drawn from $\mathscr{X} \times \mathscr{Y}$. It is clear that one will always only have direct access to the in-sample error E_{in} but we are profoundly interested in finding a regression function $f$which yields a small out-of sample error E_{out}[1]. The out of sample error can be written as

$$E_{\text{out,mse}}(f) = \mathbb{E}[(f(X) - Y)^2]$$

where the expectation is taken with respect to the probability distribution from which we draw training examples.

The concept of *generalization* deals with the question whether a small in-sample error E_{in} in a specific circumstance can be used to draw the desired conclusion that the out-of sample error E_{out} also will be small.

Another problem which arises in the field of learning is the case of *overfitting*. This problem arises primarily when the training data contains noise. Given a hypothesis-set which is "big enough" and that we have a learning algorithm which tries to train the model by minimizing the in-sample error, it might be the case that we will learn the noise in the given training data. This will result in a worsened out-of-sample performance for the proposed model f_{reg}. One way to circumvent this problem is that we partition the given data in to *training*, *validation* and *testing* data. The general idea is to train the model on the training data and using the validation data to break the training if the error on this set starts to grow. The testing data is used to assess the models out-of-sample performance.

We summarize these initial remarks with the following list of attributes one needs to consider in the training process.

- Choice of structural form of f_{reg}, i.e. choosing the hypothesis set $\mathscr{H}$.
- Choice of error measure in order to quantify the goodness of fit.

[1]It is intuitively clear that for the in-sample-error of a regression model to be an indicator (used in creating an upper bound) of the out-of-sample performance the distribution by which the training examples where generated must be close to the distribution that is used for out-of-sample evaluation.

- Choice of learning algorithm.
- Choice of over-fitting stopping criteria.

4 Tools for Solving

4.1 Feed Forward Artificial Neural Networks and Back-propagation

One class of regression models are the so called Feed Forward Artificial Neural Networks (FFANN). Consider the learning problem with $\mathscr{X} = \mathbb{R}^d$ and $\mathscr{Y} = \mathbb{R}^\ell$. An FFANN is characterized by its hyper-parameters $h_i \in \mathbb{N}$ for $i = 1, \ldots, L$ where L is the number of *hidden layers* of the model. We define $h_0 = d$ and $h_{L+1} = \ell$ (corresponding to the input and output layer respectively). To each $i = 0, \ldots, L$ we let

$$W_i \in \mathbb{R}^{h_i \times h_{i+1}}$$

be a weight matrix. We also introduce the functions $\sigma_i : \mathbb{R} \to \mathbb{R}$ and vectors $\mathbf{b}_i \in \mathbb{R}^{h_i}$. Given $\mathbf{x} \in \mathbb{R}^d$ in the input space we define

$$\mathbf{x}_{i+1} = \sigma_i(W_i \mathbf{x}_i + \mathbf{b}_i) \text{ for } i = 1, \ldots, L \text{ with } \mathbf{x}_0 = \mathbf{x}.$$

The mapping

$$\mathbf{x} \mapsto \mathbf{x}_{L+1}$$

is our FFANN

$$f_{\text{reg}}(\mathbf{x}) = f_{\text{reg}}(\mathbf{x}; h, W) := \mathbf{x}_{L+1}.$$

A graphical illustration of a NN is shown in Figure 10. Each $\mathbf{x}_i$ is called a *layer*, the FFANN operates on a given layer $\mathbf{x}_i$ yielding the layer $\mathbf{x}_{i+1}$ by multiplying $\mathbf{x}_i$ with the corresponding weight matrix adding the bias and applying the *activation function* σ_i to each component of the vector $W_i \mathbf{x}_i + \mathbf{b}_i$. There are many possible choices of activation functions but one standard choice is

$$\sigma_i(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} = \frac{e^{2s} - 1}{e^{2s} + 1} = 1 - \frac{2}{e^{2s} + 1} \tag{3}$$

for all $i = 1, \ldots, L$ (which all are smooth function tending to 1 as $s \to \infty$ and to -1 as $s \to \infty$) and furthermore to take $\sigma_{L+1} = \text{Id}$ to be the identity map.

Using the mean squared error as the quantification of the error we train our model by choosing hyper-parameters h_i and L and then solving the minimization problem

$$\min_{W,b} E_{\text{in}}(W, b). \tag{4}$$

where

$$E_{\text{in}}(W, h) = \frac{1}{N} \sum_{n=1}^{N} \| f_{\text{reg}}(\mathbf{x}_n) - \mathbf{y}_n \|_2^2. \tag{5}$$

where W here denotes all the components of the weight matrices and b denotes all bias-values and $(\mathbf{x}, \mathbf{y})$ the given training data. This is in general a non-convex optimization problem. One standard method of finding a local minimum of $E_{\text{in}}(W, h)$ is to use gradient descent where $(W_{n+1}, b_{n+1}) = (W_n, b_n) - \eta \nabla_{(W,h)} E_{\text{in}}(W_n, h_n)$. The parameter $\eta > 0$ is called the *learning rate*. There is an efficient process by which the gradient $\nabla_{(W,h)} E_{\text{in}}(W_n, h_n)$ can be computed which is called *back propagation*.

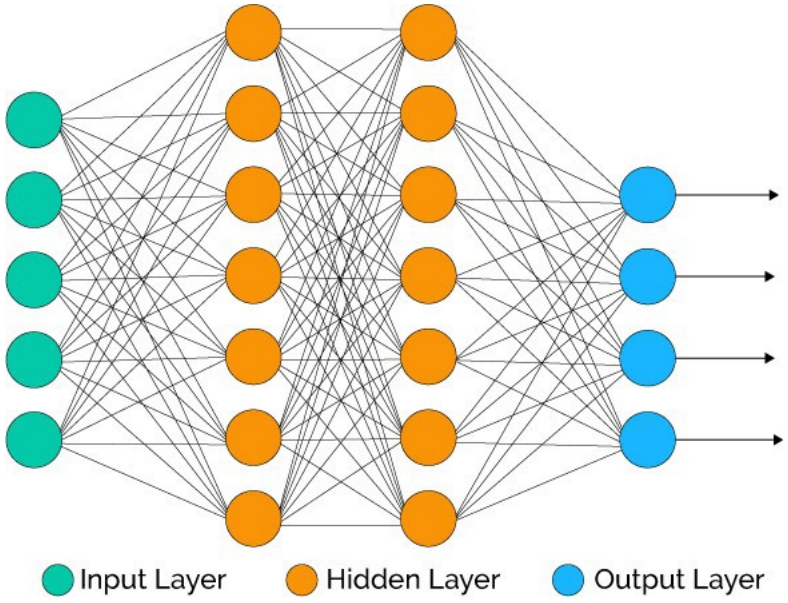

Figure 10: Graphical illustration of a Feed Forward Artificial Neural Network. The figure shows that one node in the network is influenced directly by all nodes in the previous layer according $x_{i+1}^{\ell} = \sigma_i(\sum_{j=1}^{h_i}[W_i]_j^{\ell} x_i^j + b^{\ell})$

.

4.2 Regression Trees

A regression tree is a binary tree T. Suppose that the output is one dimensional so that we have $(\mathbf{x}_n, y_n)$. The leaves of the tree will contain some of the data points. Each branch leading up to one of the leaves will be associated with a condition of the type: if $\mathbf{x}^i < v^*$, (for some value v^* to be learned and index i) go left; otherwise go right, see Figure 11 for a visualization. The component $\mathbf{x}^i$ is said to be a *predictor* for the given node.

We let

$$m_c = \frac{1}{N_c} \sum_{i \in c} y_i$$

which is the prediction if the decision tree T puts us in leaf c. We then define the sum of squared errors for a tree T to be

$$S = \sum_{c \in \text{leaves}(T)} \sum_{i \in c} (y_i - m_c)^2.$$

The regression-tree-growing algorithm works as follwos: put all training data in one node of the tree and calculate m_c and S. In the growth step we search over all binary splits of the data which will reduce S as much as possible. This split is computed with respect to one of the components of the input vector $\mathbf{x}$, say $\mathbf{x}^i$, which yields a rule in the tree in, as explained above. The choice of which of the input variables to do the split over (i.e. choice of index i) can be random; or one chooses in the algorithm which of the input variables gives the largest reduction in S. In the case that the component $\mathbf{x}^i$ that we consider is an ordered component and our data set has n-distinct values of this component we only consider the $n-1$ different splittings which gives us two nodes where the ordering is kept and a corresponding rule: go to left child if $x_1 < v^*$ and go to right child if $x_1 \geq v^*$.

When such a partition is found, repeat the growing process for each leaf of the current tree.

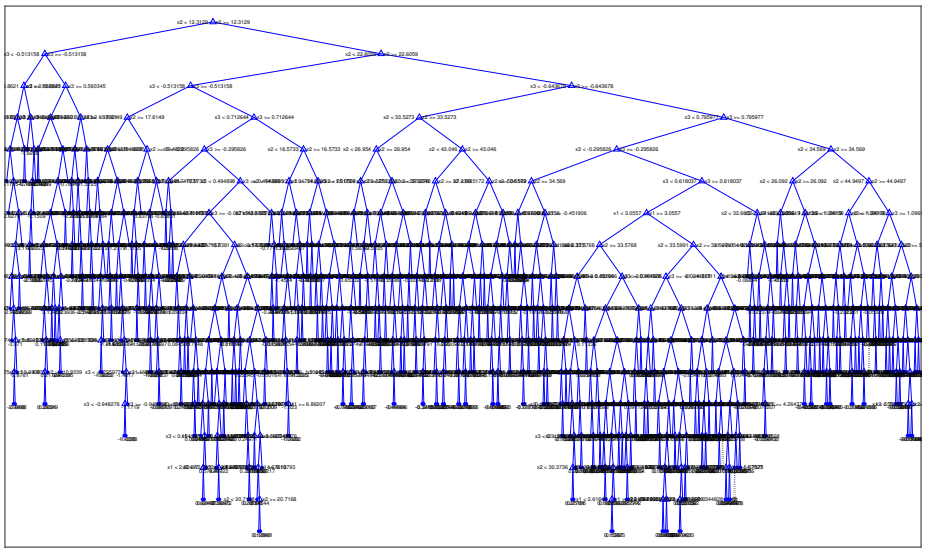

Figure 11: Graphical illustration of a regression tree for the given data set.

If this process is not stopped there is a possibility is that the tree will end up having the same amount of leaves as the size of the training set, that is one data point in each leaf. In order for this not to happen we stop the growing process if a binary split would result in one of the child-leafs would have less than q-points (for some fixed parameter q). Furthermore we also stop the growing process if the decrease in S would be less than some pre-defined $\delta > 0$. Finally we stop the growing process if all elements of a leaf have the same input parameters $\mathbf{x}_n$. This would happen in the case that the data we are trying to fit does not represent a mathematical function (in the strict sense that one input yields only one output).

Another technique used in order to combat the effects of overfitting is that a regression tree can be *pruned*, meaning that subtrees are removed and replaced by leafs instead. Therefore, some algorithms used grow a tree so that each node contains very few nodes, and then starts pruning to reduce the complexity of the final regression tree.

In Matlab a regression tree can be built using the `fitrtree` -command.

4.3 Random Forest

An individual regression tree tends to overfit the data. A regression forest grows many trees and use the combined result of these trees to predict the data. This can be achieved using the `TreeBagger`-command in Matlab.

Bagging stands for *bootstrap aggregation* which means that each tree in the the forest is trained on a *bootstrap replica* of the given data-set. This means that the data is given resampled by drawing Nout of N observations with replacement (inherently not using all available data for each of the regression trees). In order to assure that the trees in the forest are not to similar some predictors for some of the nodes are chosen in a random fashion.

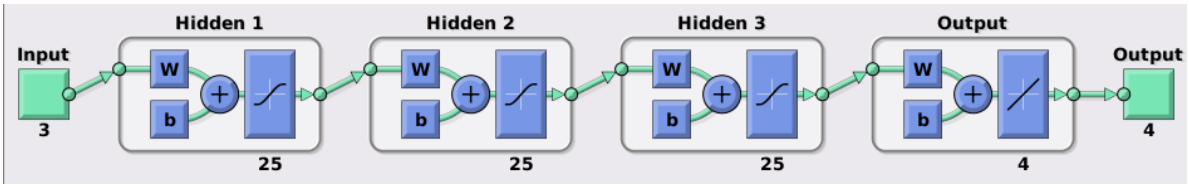

Figure 12: Graphical illustration of the neural network of 3 hidden layers with 25 neurons per layer which is used for Task 1.

5 Task 1: Fitting System Model

In order to address this task, we experimentate with different algorithms that have been presented in the previous section in order to fit the given input-ouput data. The procedure and the parameters chosen is explicitly described. Comparison between methods as well as experimental results are provided in detail.

5.1 Experimental Results

The first method to be investigated was a feed-forward neural network with sigmoid base functions as in (3). Neural Network Toolbox of MATLAB is used which can implement three algorithms:

- Levenberg-Marquardt;
- Bayesian Regularization;
- Scaled Conjugated.

By trying different experiments, the first algorithm was performing better to our given data. Initially, the MIMO implementation was chosen, i.e., an architecture with 3 inputs and 4 outputs. The neural network architecture is 3 hidden layers with 25 neurons each, as is depicted in Figure 19. The outcome of the training is a function $\hat{f}(\cdot)$. Then, we compare the given output y with the output of $\hat{y} = \hat{f}(u)$, where u is the given input data. Due to the fact that some values of $\hat{y}$ are close to zero, instead of computing the relative error, we used the robust relative error, as defined in (1).

Method	y_1		y_2	
	Mean	Var	Mean	Var
1NN	$2.4{\cdot}10^{-3}$	$1.6{\cdot}10^{-2}$	$1.7{\cdot}10^{-3}$	$3.5{\cdot}10^{-3}$

Table 1: Mean and variance of robust relative errors of the samples for output y_1 and y_2.

Method	y_3		y_4	
	Mean	Var	Mean	Var
1NN	$7.1{\cdot}10^{-4}$	$2.8{\cdot}10^{-3}$	$3.4{\cdot}10^{-5}$	$3.7{\cdot}10^{-5}$

Table 2: Mean and variance of robust relative errors of the samples for output y_3 and y_4.

Table 1 and 2 shows the mean and the variance of the robust relative errors of every sample. The total training procedure takes 7 hours in a computer with 16GB of RAM

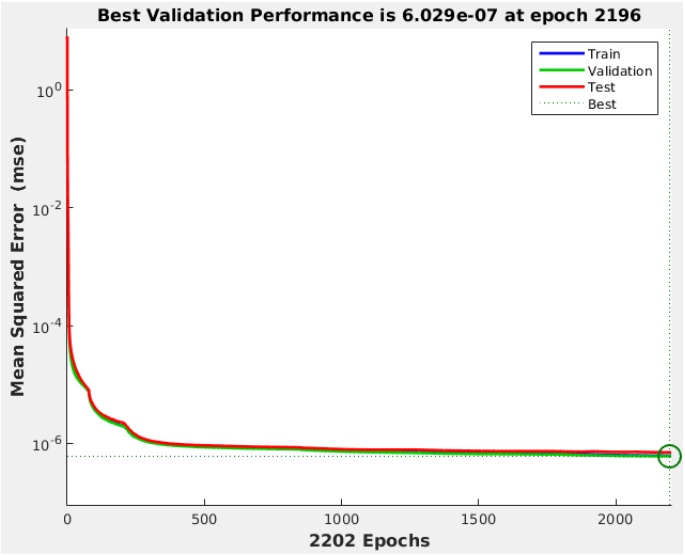

Figure 13: Graphical illustration of the convergence rate of training and validadtion data.

and i7 CPU with frequency 3.6GHz. The convergence rate of training and validation data is depicted in Figure 13. In the next subsection, we compare these results with other statistical methods/algorithms.

5.2 Experiments with other Methods/Algorithms

After having experimentated with a MIMO neural network of 3 hidden layers and 25 neurons per layer, we compare these results with the following algorithms/methods:

1. 4 Multiple Input-Multiple Output (MISO) models;
2. 1 Regression Tree (RT);
3. Random Forest Algorithm (RFA) with multiple trees.

Methods (1) were implemented in Neural Network Toolbox of Matlab; Methods (2) and (3) was implemented in Regression Learner Toolbox of Matlab. Tables 3 and 4 show the mean and the variance of the robust relative errors, as defined in (1), of all samples with reference to outputs y_1, y_2 and y_3, y_4, respectively. The outcome of all aforementioned methods is represented in different lines. It can be observed that y_1 is the most difficult to be predicted and y_4 could be predicted with high accuracy. Figure 14 shows the prediction of 2 of the methods (1 MIMO neural network and Random Forest Algorithm) for a random choice of samples.

Method	y_1		y_2	
	Mean	Var	Mean	Var
1NN	$2.4{\cdot}10^{-3}$	$1.6{\cdot}10^{-2}$	$1.7{\cdot}10^{-3}$	$3.5{\cdot}10^{-3}$
4NN	$7.4{\cdot}10^{-3}$	$3.4{\cdot}10^{-2}$	$4.5{\cdot}10^{-3}$	$5.4{\cdot}10^{-3}$
RFA	$5.1{\cdot}10^{-2}$	$8.8{\cdot}10^{-2}$	$-1.0{\cdot}10^{-2}$	$8.6{\cdot}10^{-2}$
RT	$4.4{\cdot}10^{-3}$	$5.2{\cdot}10^{-2}$	$3.7{\cdot}10^{-3}$	$3.8{\cdot}10^{-2}$

Table 3: The mean and variance of robust relative errors of the samples for different methods with reference to output y_1 and y_2.

Method	y_3		y_4	
	Mean	Var	Mean	Var
1NN	$7.1{\cdot}10^{-4}$	$2.8{\cdot}10^{-3}$	$3.4{\cdot}10^{-5}$	$3.7{\cdot}10^{-5}$
4NN	$3.8\ {\cdot}10^{-3}$	$6.5\ {\cdot}10^{-3}$	$7.3\ {\cdot}10^{-4}$	$7.2\ {\cdot}10^{-4}$
RFA	$2.6{\cdot}10^{-2}$	$7.7{\cdot}10^{-2}$	$2.4{\cdot}10^{-2}$	$7.2{\cdot}10^{-3}$
RT	$2.4{\cdot}10^{-3}$	$4.5{\cdot}10^{-2}$	$4.3{\cdot}10^{-4}$	$1.6{\cdot}10^{-3}$

Table 4: The mean and variance of robust relative errors of the samples for different methods with reference to output y_3 and y_4.

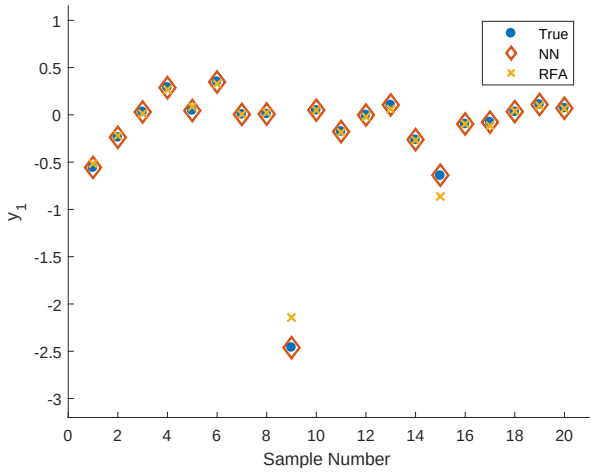

Figure 14: Graphical illustration of estimation of output y_1 by 2 of the experiments: one MIMO neural network and Random Forest Algorithm.

5.3 Comparison of Methods

After performing many experiments with the aforementioned methods we conclude that each method has advantages and disadvantages. There is a trade-off between desired specifications that a user needs to impose. We mention hereafter the conclusion statements we reached after experimentation:

- NN methods require tuning of large amount of parameters; RT method is tuning-free; RFA requires tuning only regarding the number of trees.
- NN require big amount of computation in order to train the network properly; RT and RFA methods are relatively fast.
- NN requires pre-scaling of the data, while RFA and RT function without any pre-scaling.
- NN returns as output a function which can be processed fast with small storage space RT and RFA return output data structures which might be slow at process and require large amount of system memory.
- NN are MIMO compatible while RFA and RT are not.
- All methods are robust in noisy changes of initial given data.

The aforementioned statements are summarized in Table 5.

Criteria	**NN**	**RFA**	**RT**
Tuning hyper-parameters	a lot	low	low
Speed of training	7 h	7 min	1s
Pre-processing (scaling)	needed	no	no
Speed of function evaluation	fast	slow	slow
MIMO compatible	yes	no	no
Robustness (input-output)	yes	yes	yes
Model Information Storage	small	big	big

Table 5: Comparison of different methods/algorithms with respect to desired designer's specifications.

It should be noted here that the aforementioned conclusions are drawn by experimental testing. Thus, general conclusions with theoretical guarantees could not be provided. An interested reader is suggested to read [1, 2] for more technical information regarding the comparison of methods/algorithms.

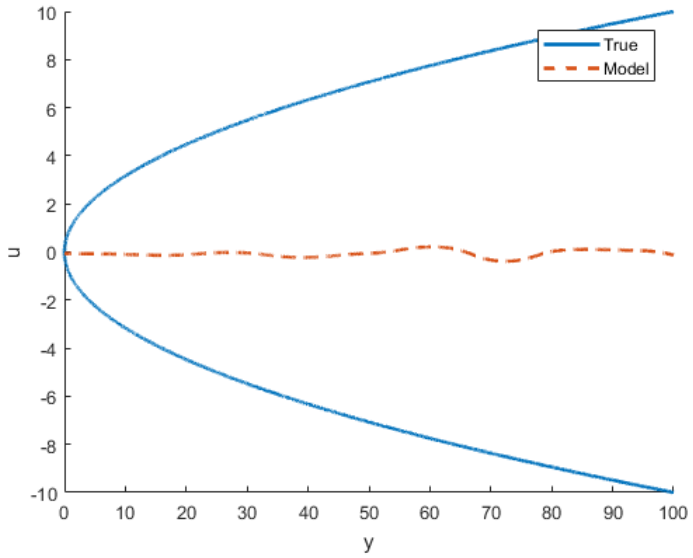

Figure 15: Figure with Example 1

6 Task 2: Fitting Inverse Model

In what follows, we consider the problem of constructing an inverse model for the system. Recall that in this problem, we are given the data-set $\{(\mathbf{u}_n, \mathbf{y}_n)\}_{n=1,\dots,N} \subset \mathbb{R}^{3\times 4}$, and we would like to find a regression model $f_{\text{reg inv}}$ such that $\mathbf{u} \approx f_{\text{reg inv}}(\mathbf{y})$ in some appropriate sense. In this section, we propose and investigate several methods for constructing an inverse model of the system. Prior to that, we explain difficulties we may encounter with the inverse models. We also propose a possible approach that can be used in the case that inverse models of desired quality cannot be obtained.

6.1 Difficulties with Inverse Model

On the first glance, one may assume that the problem of estimating an inverse model does not differ from the problem of estimating the original mapping. We can simply use the outputs of the system as an input to the machine learning algorithm, while the inputs should be outputs of the algorithm that we want to approximate. However, this problem may introduce additional difficulties, which we try to illustrate on the following example.

Example 1. In this example, we consider a problem of fitting inverse of quadratic function $y = x^2$ on the interval $[-10, 10]$. Note that for this interval, two different inputs u_1 and $u_2 = -u_1$ map into the same output: $y_1 = y_2 = (u_1)^2$, that is, inverse function is non-injective. Assume that a machine learning algorithm we use tries to minimize mean square error of the sample. In that case, if we give this algorithm a task to calculate a best estimate of the inverse function, the result will be zero due to symmetry, as illustrated in Figure 15.

6.2 Investigating Non-Injectivity

Motivated by the previous example, we tried to discover if our system is non-injective. However, while a function of one variable can easily be plotted and non-injectivity can

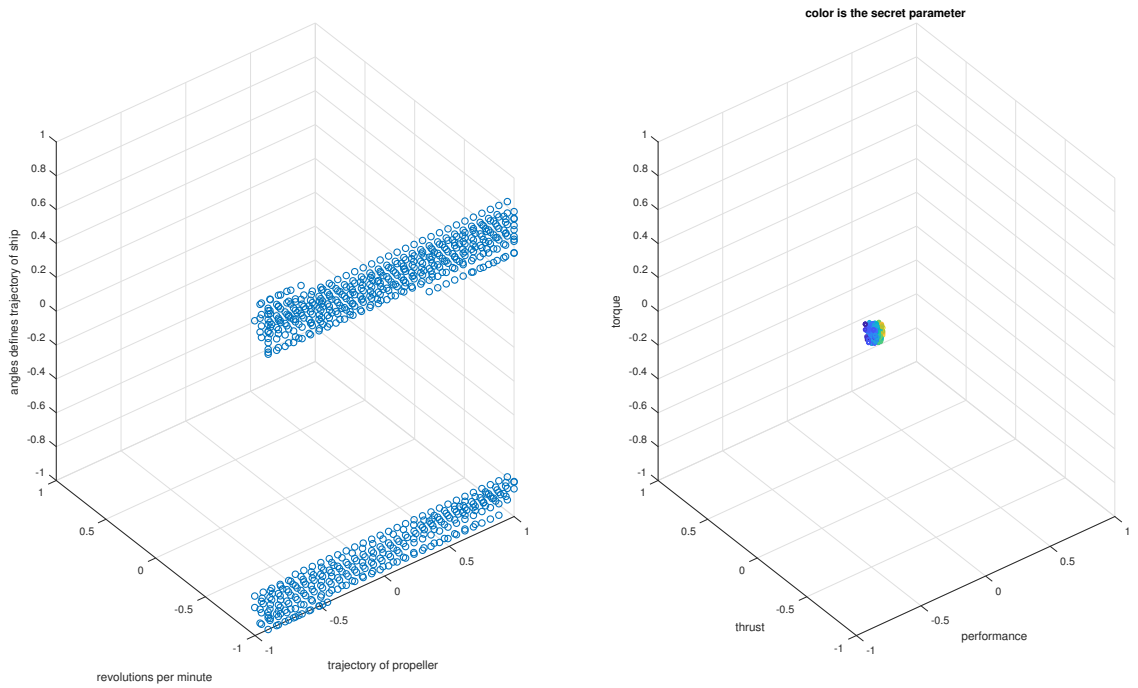

Figure 16: Figure visualizing "non-injectivity" of our mapping.

	u_1		u_2		u_3	
	Mean	Var	Mean	Var	Mean	Var
3NN	$4.2{\cdot}10^{-2}$	$5.8{\cdot}10^{-2}$	$9.2{\cdot}10^{-4}$	$2.0{\cdot}10^{-4}$	$-7.5{\cdot}10^{-3}$	$5.7{\cdot}10^{-2}$
RFE	$1.8{\cdot}10^{-2}$	$7.8{\cdot}10^{-3}$	$2.3{\cdot}10^{-4}$	$1.5{\cdot}10^{-4}$	$-8.2{\cdot}10^{-3}$	$2.4{\cdot}10^{-3}$

be investigated by observing the plot, investigating this property in higher dimensional spaces is not straight forward.

In Figure 16 the input and output spaces are visualized. The output space is 4-dimensional where the fourth dimension is represented by the color of the point.

We see that two regions of distinct inputs are mapping to the same small neighborhood (tol = $\pm 5\%$). The visualization suggests that our function is non-injective in the sense that inputs which lie in different parts of the input space are mapped to the same neighborhood of the output space.

However, we are given data drawn from an analytical model. The question of whether this model is non-injective can not be fully answered only having a finite data set. In the case two of the data points have exactly the same output but with different inputs non-injectivity can be concluded.

6.3 Experimental Results

To obtain inverse model of the system, we fitted 3 Multiple Input-Single Output (MISO) models using neural networks and random forests. Table 6.3 show the mean and the variance of the robust relative errors for inputs u_1–u_3, as defined in (1).As we can see, the quality of input estimates is comparable to the quality of output estimates. In this case, output u_1 was the most difficult to estimate, and the output u_2 was estimated with the highest quality.

6.4 Possible Approach for Non-Injectivity Case

In this particular case, we managed to obtain inverse models of relatively good quality. The question is however what to do once this is not the case. Here, we propose a possible approach that can be used if non-injective of a function represents an issue.

The idea is to use the original model of the system, and to formulate the problem of finding an optimal control input as an optimization problem of the following form.

$$\begin{aligned} \underset{u}{\text{maximize}} \quad & y_1 \\ \text{subject to} \quad & |y_i - y_i^*| \leq \epsilon_i, && i \in \{2,3,4\}, \\ & \underline{u}_j \leq u_j \leq \bar{u}_j, && j \in \{1,2,3\}, \\ & y = f(u). \end{aligned}$$

The objective function represents the efficiency (output 1), which we want to maximize. The first constraints imposes that other outputs should be sufficiently close to some desired reference y_i^*. The second constraints ensures that the inputs remain inside a certain predefined bounds, while the third output ensures that the outputs and inputs satify the physical model.

This problem is non-convex optimization problem in general, so we would be able to recover the exact solution of the problem only in some special cases. Nevertheless, nowadays there are solvers that work relatively well for these types of problems, for example *Gurobi*.

7 Task 3: Robustness

In this section, we investigate how robust is the estimated function $\hat{f}(\cdot)$ to small changes of given data, i.e., in the presence of measurement noise. In order to test this, noise of amplitude

$$d \cdot \text{randn}(\cdot), \text{where } d \in \left[\frac{1}{2^6}, \frac{1}{2}\right],$$

is added to the given data and we compare the output results on out of training data. Figure 17 and 18 and 19 shows the outcome of the experiments.

We see that the single regression tree shows that the relative error in the out-of-sample performance is downward trending with approximately linear order in relation to the relative error imposed on the outputs the training data-set.

We do not observe any significant reduction of the corresponding error for the regression forest algorithm, where all relative errors are of the same magnitude for all disturbances.

These findings suggests that the imposed error for the random forest is immaterial to the models predictive power, meaning that this type of model handles the input-noise in a better fashion. The single-regression-tree-model will tend to overfit the data given in the training session, hence the noise will tend to be encoded in to the model and the out-of-sample performance will be deteriorate.

We see that the trend for the FFANN is that the error is relatively independent of the noise added, which suggests that these types of models for this data set are robust.

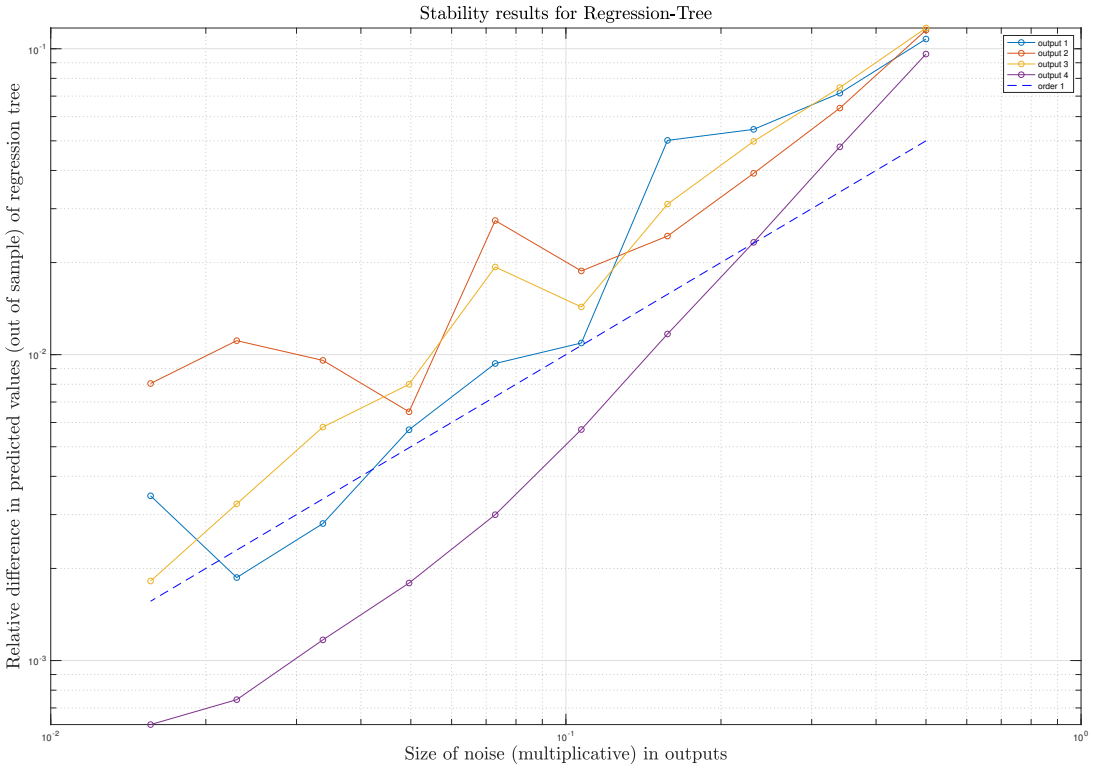

Figure 17: Graphical illustration of the relative difference in predicted values (out of sample) of a single regression tree. Number of training points $N \approx 60000$.

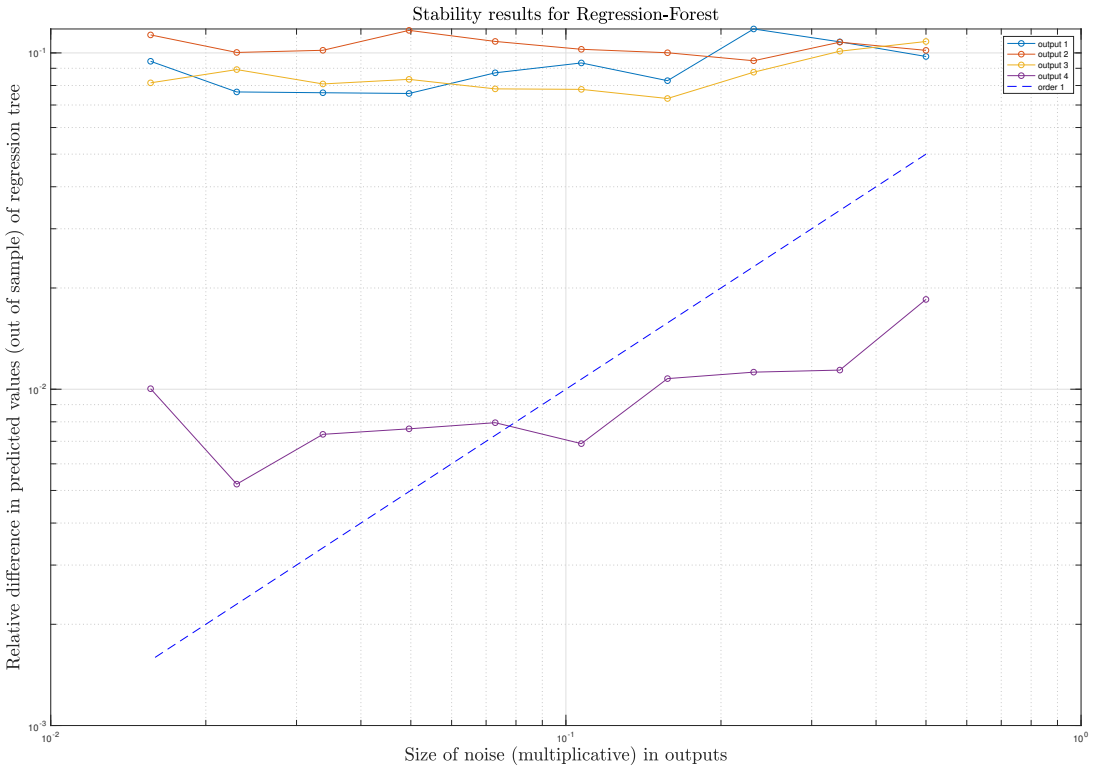

Figure 18: Graphical illustration of the relative difference in predicted values (out of sample) of a regression forest (TreeBagger in Matlab) with 10 trees, $N \approx 60000$.

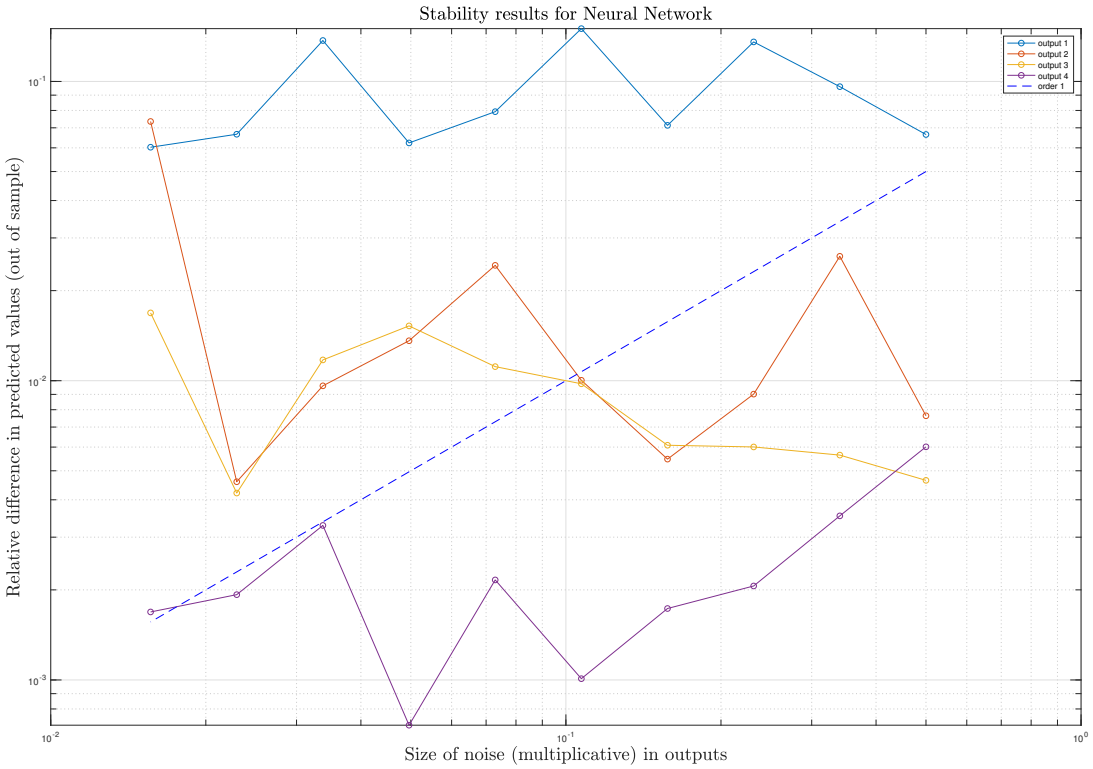

Figure 19: Graphical illustration of the relative difference in predicted values (out of sample) of a Neural Network $[5, 5, 10]$, $N \approx 6000$.

8 Conclusions

8.1 Report Summary

In this work, we cosnidered the data driven model fitting for multiple input multiple output thrust propulsion system of ABB company. In particular, the following tasks were investigated.

The first task was to create a model that maps inputs of the system to the outputs. For this purpose, we used neural networks and regression tree based methods. It was shown that both of these machine learning based methods resulted in models that predict behaviour of the output with relatively good accuracy. However, the quality of the obtained estimates differed among outputs. It turned out that the estimate of the output that was the most important one was of the poorest quality.

Secondly, we considered the problem of fitting a model that maps outputs of the system to the inputs. We firstly illustrated that difficulty of this problem may lay in non-injectivity property. We then disscussed how the non-injectivity property can be investigated in the case of the function we consider. Although we suspected that the mapping from outputs to inputs is non-injective, we managed to obtain the inverse models of similar quality as for the case for models of input–output mapping, using neural network and regression tree based methods. Finally, we proposed a solution in the case that non-injectivity represent an issue.

Finally, robustness of the model was empirically investigated. We found that the FFANN and Regression Forest was input-output robust with respect to noise in the training data.

8.2 Future Work

The future work will go into two directions. Firstly, in this work we have considered a static input-output model of the form $y = f(u)$ with data that are outcome of analytical model. Future work will be devoted towards the direction of dealing with dynamics of the thrust propulsion system, i.e., the model will be of the form:

$$\begin{aligned} x(k+1) &= f(x(k), u(k)), \\ y(k) &= h(x(k), u(k), w(k)), \end{aligned}$$

where $k \in \mathbb{N}$ is the index of samples; x the internal vector state of the system; and w stands for the output noise/disturbance. The main idea here is to is to express the current output vector $y(k)$ as a function of all the previous input and output data $u(k-1), u(k-2), \ldots$ and $y(k-1), y(k-2), \ldots$, respectively. Ideas from System Identification (SYSID) field could be employed.

Secondly, data used in this experiment was generated based on an analytical model. Thus, it did not contain any noise, and generating rich enough data set did not represent an issue. The future experiments will be based on the data collected from the real process. In that case, we expect two issues to appear:

1. The size of the data set will be limited;

2. The data will contain measurement noise and other types of noise.

References

[1] L. Breiman, "Random Forests," *Journal of Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[2] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, vol. 3. Prentice-Hall Englewood Cliffs, 1994.